Poster proposal for Hot Chips 2016

Task Parallel Programming Model + Hardware Acceleration = Performance Advantage

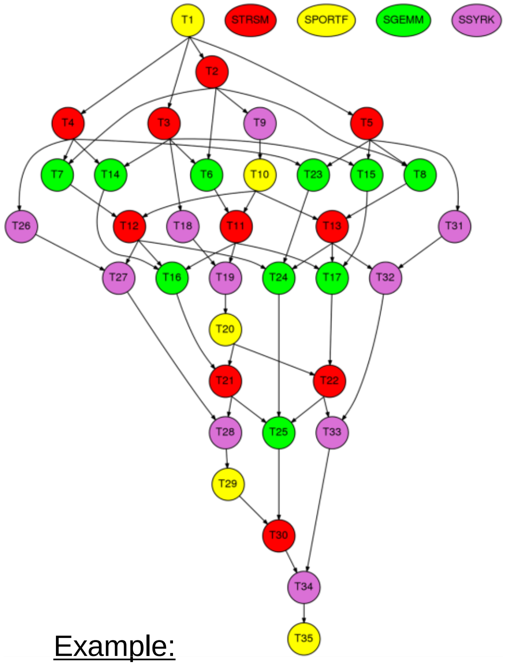
Authors:

Tamer Dallou[1], Cesar Divino Lucas[2], Guido Araujo[2], Lucas Morais[2], Eduardo Ferreira Barbosa[2], Michael Frank[3], Richard Bagley[3], Raj Sayana[3]

[1] LG Electronics Technology Center Europe, [2]University of Campinas, Brazil (Unicamp), [3]LG Electronics Mobile Research, San Jose Lab

Introduction and overview

- SoC are multi-core heterogeneous computing platforms
  - No good tools for programming (debugging, instrumentation) exist
  - writing parallel programs the old way (by using pthreads) does not scale
  - Programming for these platforms is hard and prohibits efficient use of all available compute resources
- Support many-cores and heterogeneous configurations for applications
  - We propose to adopt the TaskSuperscalar (TSs) programming model → OmpSs & OpenMP 4.0
  - Break down large programs into a connected mesh of small tasks; running on heterogeneous group of processors, use dataflow dependencies for synchronization (instead of barriers)
  - Out-of-order execution of tasks depends on inter-task dependencies
- Needs "mechanical" support - It is not feasible to have a programmer to express such a graph
  - They may be much more complicated, involving 100's of active tasks with 1000's of edges
  - Task graphs can be dynamic and change depending on data processed
- An optimizing and parallelizing compiler tool chain for future multi-core SoC and HW accelerators
  - Understand and support application optimized instruction set extensions
  - Automatically target the parallel run-time system
- Running small tasks
  - Run-time system introduces a fixed, non-negligible overhead for task creation, task submission, task issue and dependence management
  - HW support, acceleration, for critical functions, for example scheduling and dependence resolution
  - Record dataflow dependencies as tasks are submitted



<u>Example:</u>

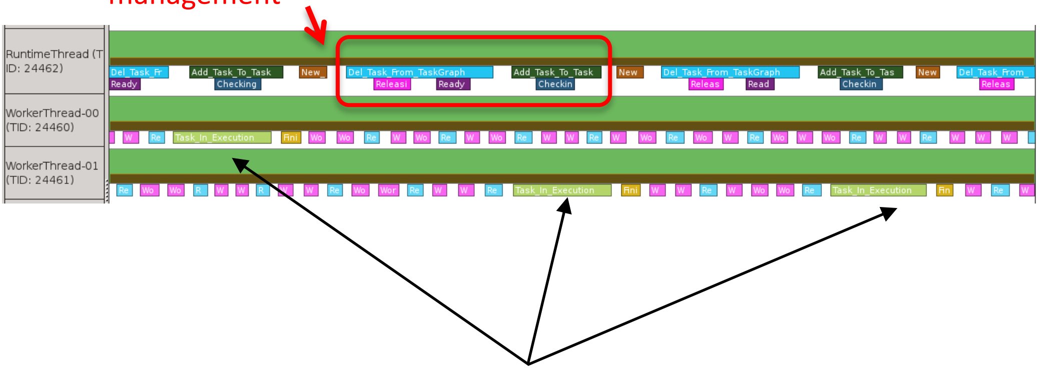
Taskgraph for Cholesky Algorithm

(35 tasks, 66 edges)

Profiling a Runtime Library for OmpSs/OpenMP

- What are the major sources of overhead?

Overhead results from runtime management



Task Size → 2,000 Add & Mul

Need to reduce runtime overhead!

→ accelerate runtime by HW

Task Info	SparseLU (-64)			
Task ID	402830	402860	4028a0	4028e0
Task cycles	246287	429212	382085	649154
Runtime cycles	9896	19457	25522	33167
#Tasks	64	1024	1024	21856

Very large tasks

Total cycles:

Runtime: ~ $7.72 * 10^8$

Work: ~ $1.50 * 10^{10}$

Overhead: ~ 5 %

Task Info	Jacobi (-32)			
Task ID	401ab0	401ea0	403040	403170
Task cycles	35736	100728	1682	6742
Runtime cycles	497	637	15234	38907
#Tasks	512	1024	4096	4096

Very small tasks

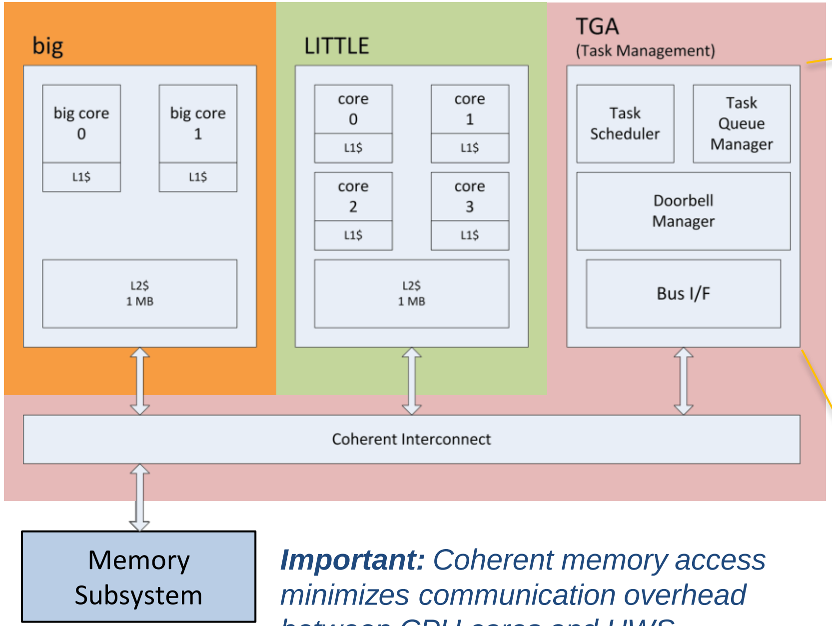
Total cycles:

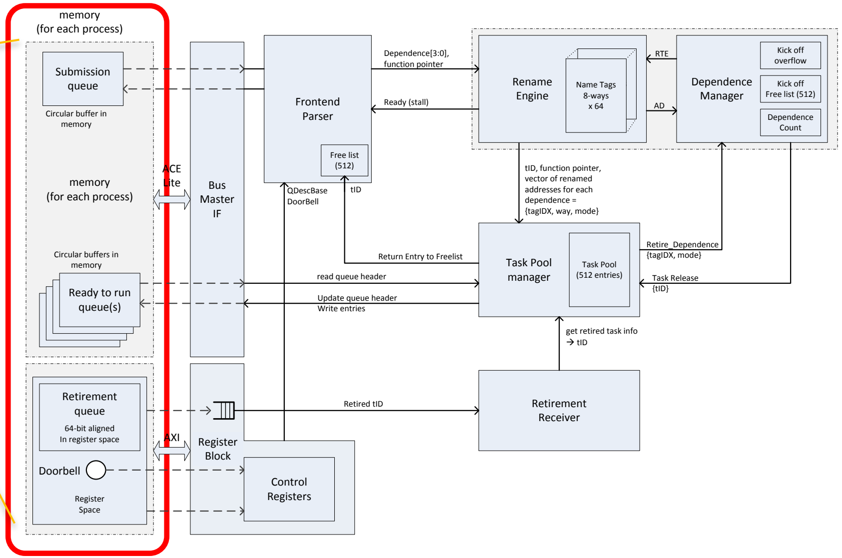
Runtime: ~ $2.23 * 10^7$

Work: ~ $1.56 * 10^7$

Overhead: ~ 143 %

Task Graph Accelerator System Level Integration





Important: *Coherent memory access minimizes communication overhead between CPU cores and HWS*

- Task Graph Accelerator key parameters
  - Estimated die area: < 0.1 mm^2 (TSMC 16FF)
  - Clock speed: > 800 MHz
  - Capacity:
    - 512 active tasks
    - 4096 graph edges (data flow dependencies) on up to 512 variables
  - Design has been implemented and demonstrated on a ZEDBOARD/ZYBO (XILINX Zynq) platform

Software Stack for Task Superscalar Programming Model

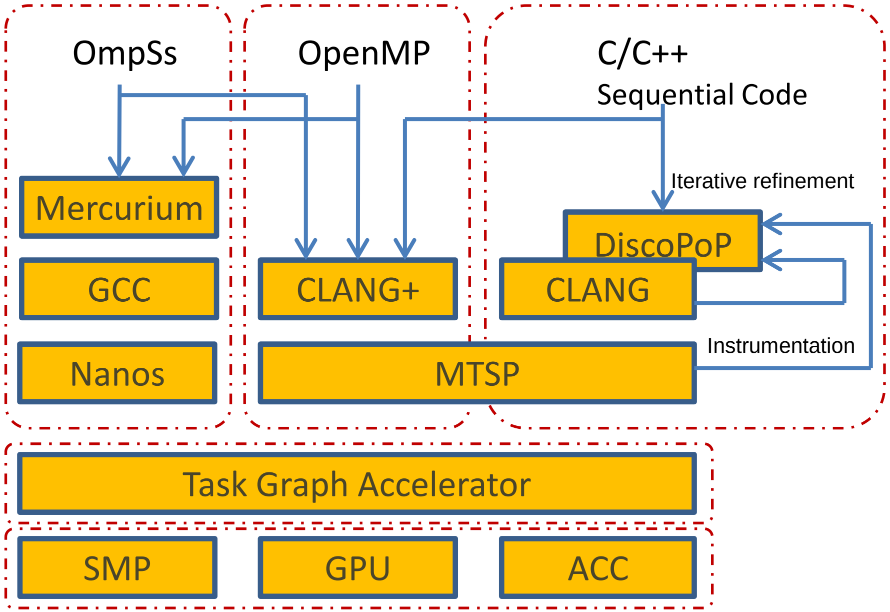
Application Level

Compiler
- *GCC = GNU compiler*
- *CLANG = LLVM compiler*
- *CLANG+ = LLVM compiler with OmpSs support added*

TSs Runtime
- *Nanos = BSC runtime*
- *MTSP = UNICAMP runtime*

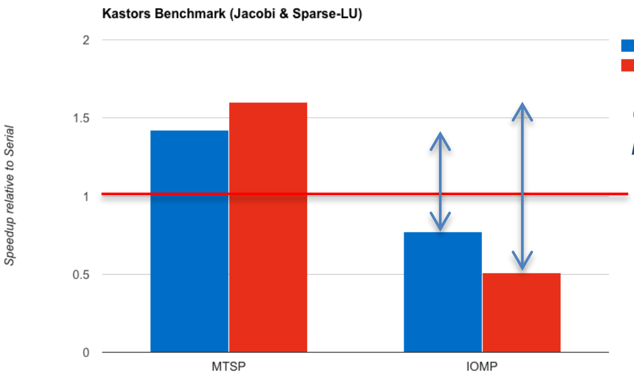
HW Runtime Accelerator

SoC Compute Platform



DiscoPop
- *Automatically parallelize*
- *Identify CU tasks (computational Units)*

TGA Performance Data



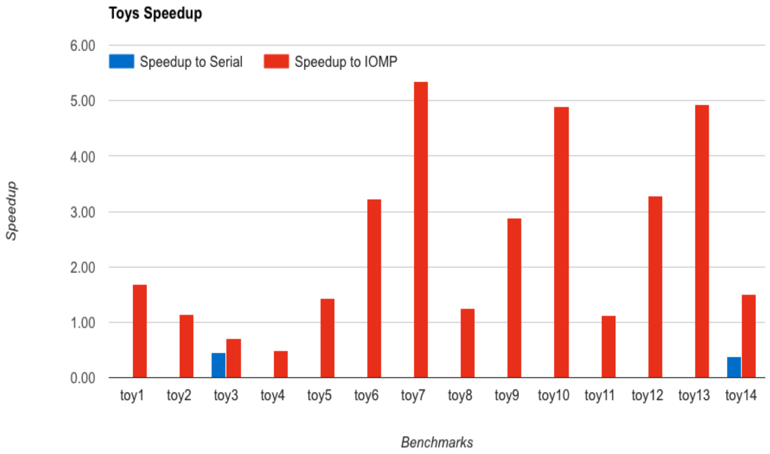
Gain = equivalent of up to 1 processor core

serial execution (1 core)

MTSP = UNICAMP runtime (with HWS)

IOMP = Intel OpenMP runtime

Both benchmarks use 2 threads



All benchmarks run on a Xilinx Zynq FPGA platform:

- 2 cores = CA9 @660 MHz; TGA running at 50 MHz; should be running on more cores to show real benefit of task graph acceleration
- With more CPU cores, the speedup over serial and IOMP will be higher) depending on the number of available cores

Kastors Benchmark (https://gforge.inria.fr/projects/kastors/)

- The KASTORS benchmarks suite was designed to evaluate OpenMP 4.0 task dependencies
- Modified state-of-the-art OpenMP 3.0 benchmarks and data-flow parallel linear algebra kernels to make use of tasks with dependencies
- KASTORS can also be used to evaluate performance of OpenMP implementations of task dependencies compared to global taskwait-based approaches

Toy Benchmarks

- Synthetic benchmarks exposing different number and type of dataflow dependencies